

CS193N
C# and the .NET Framework

Instructor: Robert Plummer, Ph.D.
plummer@cs.stanford.edu

TA: Hector Chan
chanhp@stanford.edu

June 27, 2007

Why Have a Microsoft .NET Course?

- .NET is a technology for making software easier
- .NET introduces C#
- .NET is a case study in modern framework design
- .NET is an empowering technology

Who Should Take This Course?

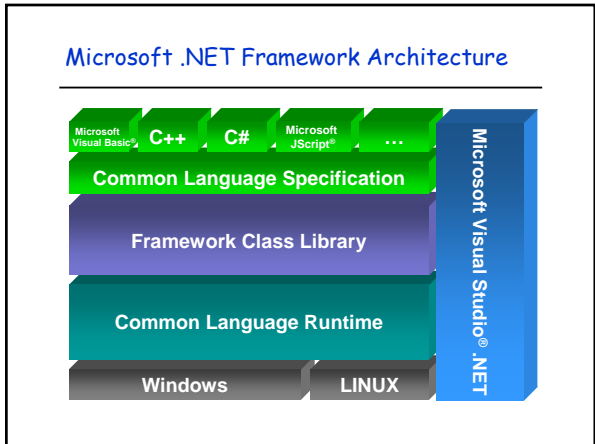
- It works for a broad audience
- The only requirement is object-oriented programming (OOP) experience

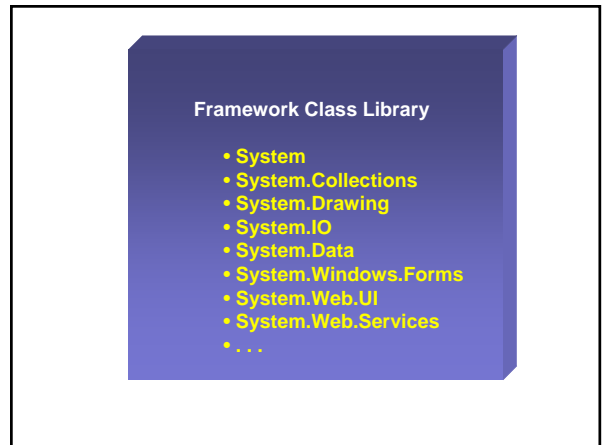
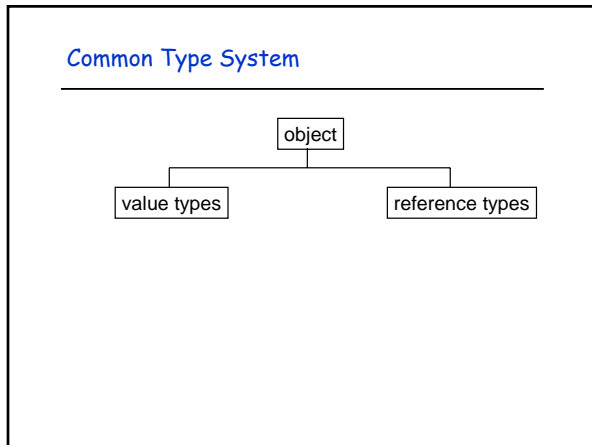
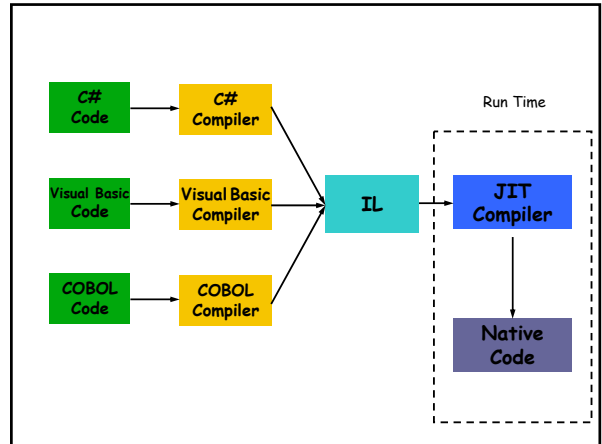
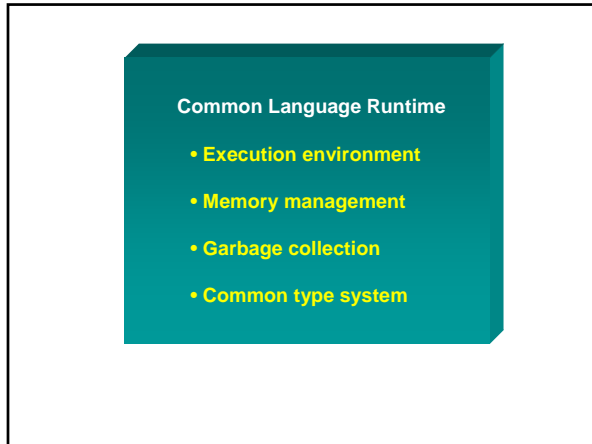
What Are the Components of the Course?

- Decision: Breadth vs. Depth
- Our approach: Breadth

What Are the Components of the Course?

- Microsoft .NET Framework overview
- C#
- WinForms
- .NET event model
- Database programming (ADO.NET)
- ASP.NET Web pages
- Web services
- Other interesting aspects of .NET
 - Threading
 - FileIO
 - Graphics
 - Cryptography
 - ...





So What Do We Get from .NET?

- A managed execution environment
- Lots of libraries
- Application programming interfaces (APIs) for writing windows- and internet-based software

So What Do We Get from .NET? (continued)

- Language interoperability
- An integrated development environment (IDE) for rapid development
- An interesting new language: *C#*
- Standards

Common Type System

- Common Language Infrastructure (ECMA Standard 335)
 - Common language specification
 - Common type system
 - Common intermediate language
- C# (ECMA Standard 334)

Object-Oriented Programming

- Abstraction
- Encapsulation
- Inheritance
- Polymorphism

C#: Why It's Important

- Designed specifically for writing .NET code
- Includes modern language features
- An improvement over existing languages

C#: Design Goals

- A simple, modern, general-purpose, object-oriented language
- Support for software engineering principles such as
 - type checking
 - array bounds checking
 - detection of use of uninitialized variables
 - garbage collection
 - multi-threading
 - exception handling
 - explicit pointers only in "unsafe" code

C#: Design Goals (continued)

- Suitable for applications deployed in a distributed environment
- Familiar syntax for C/C++ programmers
- Economical in use of memory and processing requirements, but not competing directly with C or assembly language on size or performance

C#: Language Constructs

Operators and Expressions	same as C++
Arrays	1-D, multi-D, jagged
Flow Control	if/else switch goto continue break return
Iteration	for foreach while do/while

C#: Classes

- Single inheritance
- Can implement multiple interfaces
- Members
 - Fields, methods (including constructors), properties, indexers, events
 - Access control: public, protected, internal, private
 - Static and instance members
 - Abstract and virtual methods
- Nested types
- Partial classes

C#: Getting Started

<http://cs193n.stanford.edu>

Online References
<ul style="list-style-type: none"> ■ Common Tasks Quick Reference ■ MSDN .Net SDK ■ MSDN .Net Class Libraries
Further Reading
<ul style="list-style-type: none"> ■ C# Specification Hyperlinked version ■ CLR Technical Overview ■ ECMA Standardization ECMA Standard 334 (C#) ECMA Standard 335 (CL) ■ MSDN .Net Architecture Center

C#/.NET: Some Recommended Books

[Microsoft Visual Studio 2005 UNLEASHED](#), by Lars Powers and Mike Snell
How to use VS2005 to do just about everything. (Sams)

[A Programmers Introduction to C# 2.0](#), by Eric Gunnerson & Nick Wienholt
A thorough summary of C# 2.0. (APress)

[C# 2.0 Practical Guide for Programmers](#), by Michel de Champlain and Brian Patrick
A concise, inexpensive summary of C# 2.0. (Morgan Kaufman)

[Programming Microsoft ADO.NET 2.0 Core Reference](#), by David Szeppa
Everything about accessing databases. Code in C# and VB. (MS Press)

[Pro ASP.NET 2.0 in C# 2005](#), by Matthew Macdonald and Mario Szpuszta
Thorough, readable coverage of Web Forms and Web Services. (APress)

[Programming ASP.NET 2.0 Core Reference 2005 Edition](#), by Dino Esposito
Another good book on ASP.NET (MS Press)

C#/.NET: Some Recommended Books

[Programming Microsoft ADO.NET 2.0 Applications: Advanced Topics 2005 Ed.](#), by Glenn Johnson
Covers the finer details of ADO.NET 2.0. (MS Press)

[Data Binding with Windows Forms 2.0](#), by Brian Noyes
Everything about data binding. (Addison-Wesley)

[CLR via C#](#), by Jeffrey Richter
THE book for a low-level understanding of topics like garbage collection. A little too detailed for this class, but if you become a serious .NET programmer, you will want this book. (MS Press)

[Programming Microsoft Windows with C#](#), by Charles Petzold
The C# version of Petzold's classic. Does not use Visual Studio. (MS Press)

Relating C# to Other Languages

- "Doing this in C# is just like doing it in _____"
- There are plenty of good books with titles like "C# for _____ Programmers"

[From Java to C#, A Developer's Guide](#), by Heng Ngee Mok

[C# for Java Developers](#), by Allen Jones and Allen Freeman

Namespace - types (classes, structs, enums, interfaces, delegates)
- other namespaces

```

HelloProg.cs
using System;

class Hello
{
    static void Main()
    {
        Console.WriteLine("Hello, world!");
    }
}
    
```

Console is a class defined in the System namespace
WriteLine is one of its static methods
We could also write System.Console.WriteLine

This file defines a single class, called `Hello`

Any file name will do

```

HelloProg.cs

using System;
class Hello
{
    static void Main()
    {
        Console.WriteLine("Hello, world!");
    }
}
    
```

A method called `Main` is the entry point for an application

CS193N: Course Organization

Exams	
Intra-quarter Quiz	10%
Final	30%
Assignments	60%
1 short (0.75)	
3 normal (1.0)	

Pair Programming

Optional

Choose your own partner

What it means

- Two people at one machine
- One drives, one navigates
- Change roles periodically

What it doesn't mean

- "You code half, I'll code half"

Pair Programming

What it takes to make it work

- Compatible partners
- Compatible schedules
- Roughly equal abilities

What it accomplishes

- You get stuck far less often
- You help each other learn

If one person does most of the work...

- **the other one fails the exams**

Pair Programming

You can switch partners between assignments

You may work solo if you wish.

Assignments are designed so that one person can complete them in the allotted time.

Submission guidelines will be forthcoming

Honor Code

Everything you submit must be your own work or the work of your pair

Sharing between pairs is not allowed

Pair members must share the work and the roles equally

Submissions will be scanned to detect improper collaboration

Course Outline

C# Fundamentals

Developing Windows programs using WinForms

Creating Windows controls

The .NET event model

Windows graphics using GDI+

Data access using ADO.NET

Software for the internet using ASP.NET and Web Services

Advanced C# concepts

Remaining topics for today:

- C# Example
- Common type system
- Collection types
- Boxing and Unboxing

```
class Rational
{
    private int num, den;

    public Rational(int numerator, int denominator)
    {
        num = numerator;
        den = denominator;
        ReduceToLowestTerms();
    }
}
```

```
class Rational
{
    private int num, den;

    public Rational(int num, int den)
    {
        this.num = num;
        this.den = den;
        ReduceToLowestTerms();
    }
}
```

```
class Rational
{
    private int num, den;

    public Rational(int num, int den)
    {
        this.num = num;
        this.den = den;
        ReduceToLowestTerms();
    }

    public Rational(int num) : this(num, 1)
    {
    }

    public Rational() : this(0, 1)
    {
    }
}
```

Note the syntax for having one constructor call another

$$\frac{a}{b} + \frac{c}{d} = \frac{ad + bc}{bd}$$

```
class Rational
{
    private int num, den;

    .
    .

    public Rational Add(Rational r)
    {
        Rational sum = new Rational(r.num * this.den +
                                    r.den * this.num,
                                    r.den * this.den);

        return sum;
    }
}
```

```

class Rational
{
    private int num, den;

    .
    .

    public Rational Add(Rational r)
    {
        Rational sum = new Rational(r.num * this.den +
                                   r.den * this.num,
                                   r.den * this.den);

        return sum;
    }

    public static Rational operator +(Rational r1, Rational r2)
    {
        return r1.Add(r2);
    }
}

```

```

Rational x = new Rational(6);
Rational y = new Rational(23, 47);
Rational sum;

sum = x + y;

```

```

    public override string ToString()
    {
        return "[" + num + " / " + den + "];"
    }

    private void ReduceToLowestTerms()
    {
        private int g = Gcd(num, den);

        num /= g;
        den /= g;
    }

    private int Gcd(int x, int y)
    {
        if (y == 0)
            return x;
        else
            return Gcd(y, x % y);
    }
}

```

```

using System;

class RationalTest
{
    static void Main()
    {
        Rational x = new Rational(6);
        Rational y = new Rational(23, 47);
        Rational sum;

        sum = x + y;
        Console.WriteLine("The sum is " + sum.ToString());
    }
}

class Rational
{
    . . .
}

```

```

class Rational
{
    private int num, den;
    private static int howMany = 0;

    public Rational(int num, int den)
    {
        this.num = num;
        this.den = den;
        ReduceToLowestTerms();
        howMany++;
    }

    static public int NumCreated()
    {
        return howMany;
    }
}

```

```

Rational rat3 = new Rational(6);
Rational rat4 = new Rational(23, 47);
Rational theSum;

Console.WriteLine(Rational.NumCreated() + " created");

```

A common coding convention

```
class Rational
{
    private int _num, _den;
    public Rational(int num, int den)
    {
        _num = num;
        _den = den;
        ReduceToLowestTerms();
    }

    private void ReduceToLowestTerms()
    {
        private int g = Gcd(_num, _den);

        _num /= g;
        _den /= g;
    }
}
```

Another common coding convention

```
class Rational
{
    private int m_num, m_den;
    public Rational(int num, int den)
    {
        m_num = num;
        m_den = den;
        ReduceToLowestTerms();
    }

    private void ReduceToLowestTerms()
    {
        private int g = Gcd(m_num, m_den);

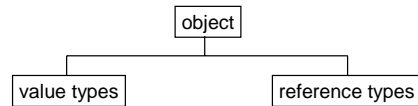
        m_num /= g;
        m_den /= g;
    }
}
```

Statements and Expressions

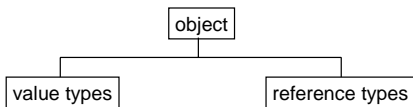
- Very similar to C++, with some changes to increase robustness
 - No '-' or ':'; all qualification uses '.'
 - Local variables must be initialized before use
 - if, while, do require bool condition
 - goto can't jump into blocks
 - switch statement - no fall through (empty case OK)
 - case labels can be strings

```
void Foo()
{
    if (i = 1) // error
        ...
}
```

Common Type System



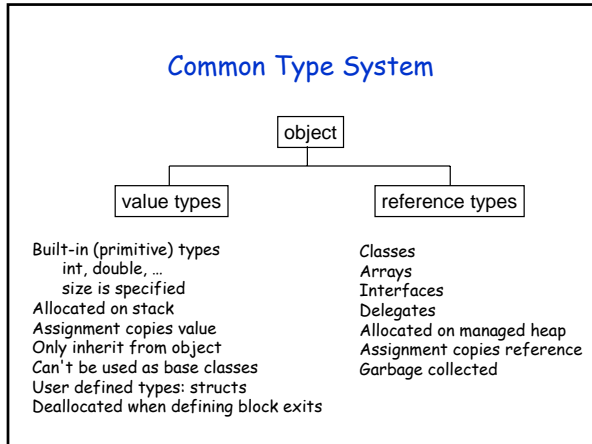
Common Type System



- Built-in (primitive) types
 - int, double, ...
 - size is specified
- Allocated on stack
- Assignment copies value
- Only inherit from object
- Can't be used as base classes
- User defined types: structs
- Deallocated when defining block exits

Types in C# and the Framework Class Library

C# Primitive Type	FCL Type	CLS Compliant
sbyte	System.SByte	No
byte	System.Byte	Yes
short	System.Int16	Yes
ushort	System.UInt16	No
int	System.Int32	Yes
uint	System.UInt32	No
long	System.Int64	Yes
ulong	System.UInt64	No
char	System.Char	Yes
float	System.Single	Yes
double	System.Double	Yes
bool	System.Boolean	Yes
decimal	System.Decimal	Yes
object	System.Object	Yes
string	System.String	Yes

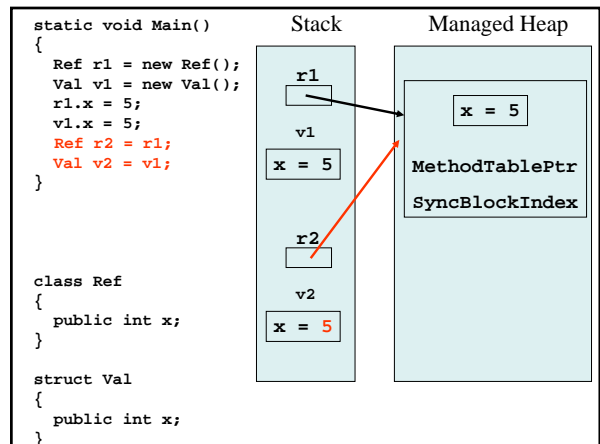
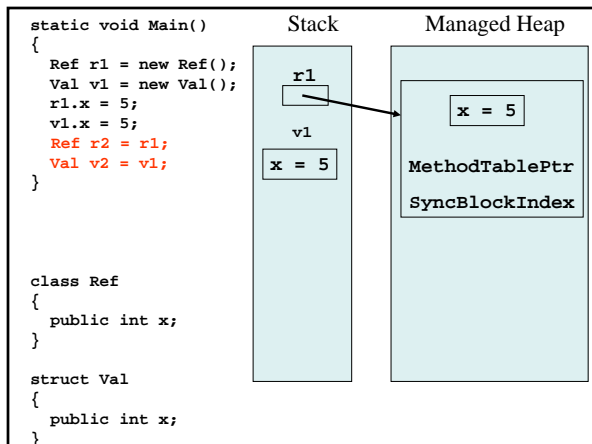
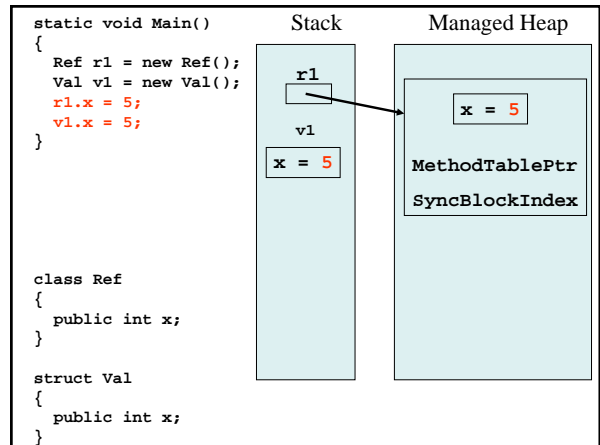
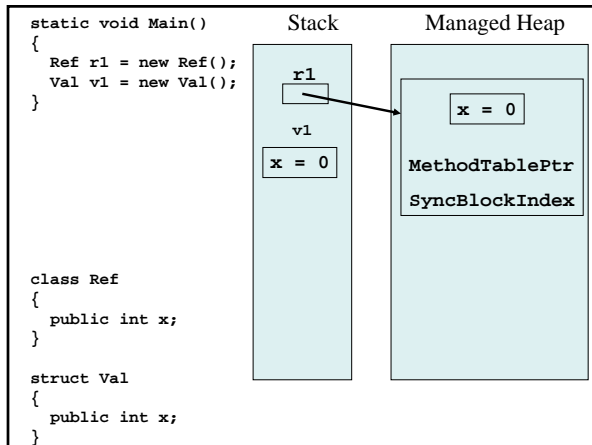


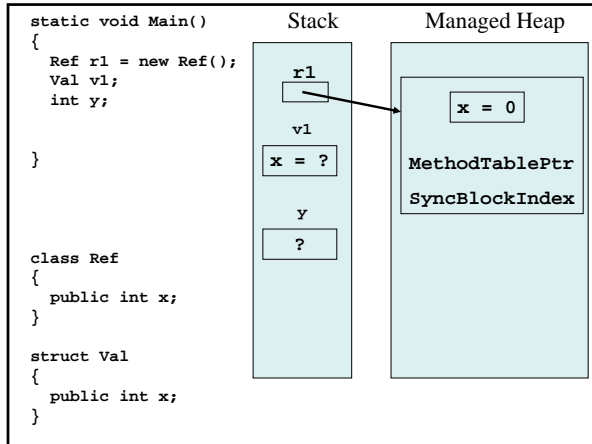
```

static void Main()
{
    Ref r1 = new Ref();
    Val v1 = new Val();
}

class Ref
{
    public int x;
}

struct Val
{
    public int x;
}
    
```





C#: Default Values

The following are automatically initialized to their default values:

- Static variables
- Instance variables of class instances
- Array elements

For **value** types, the default value is **0**

For **reference** types, the default value is **null**

Note that local variables are not considered to be initially assigned.

C# : Built-in Classes

String
ArrayList
Stack
Queue
Hashtable

These are found in the namespace

System.Collections

Some ArrayList methods

```

ArrayList.Add
  .AddRange
  .BinarySearch
  .Clear
  .Contains
  .GetRange
  .Insert
  .InsertRange
  .Remove
  .RemoveAt
  .RemoveRange
  .Reverse
  .Sort
    
```

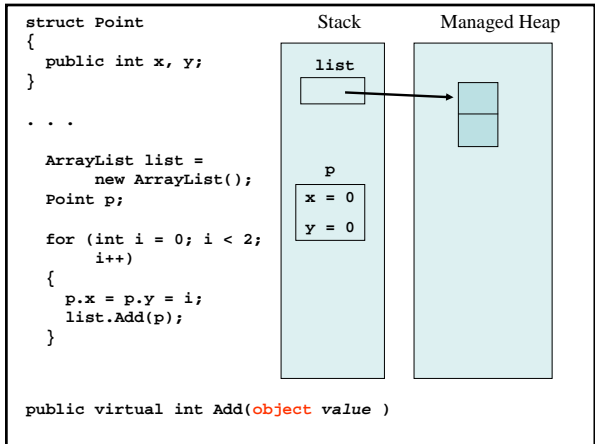
```

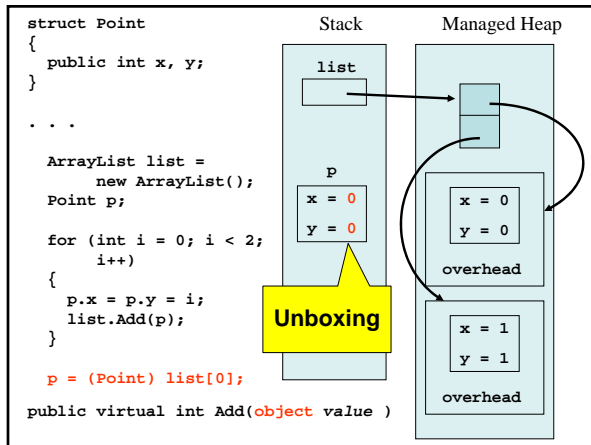
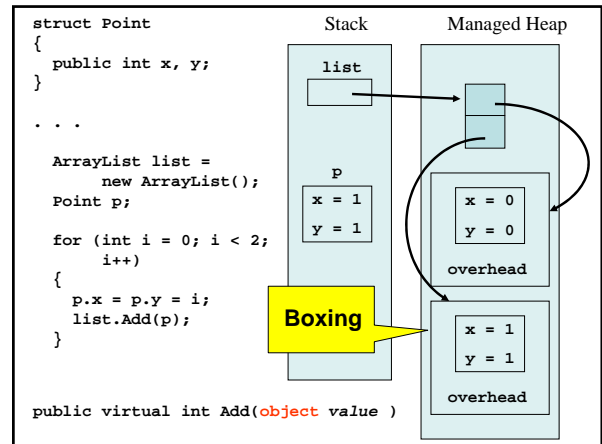
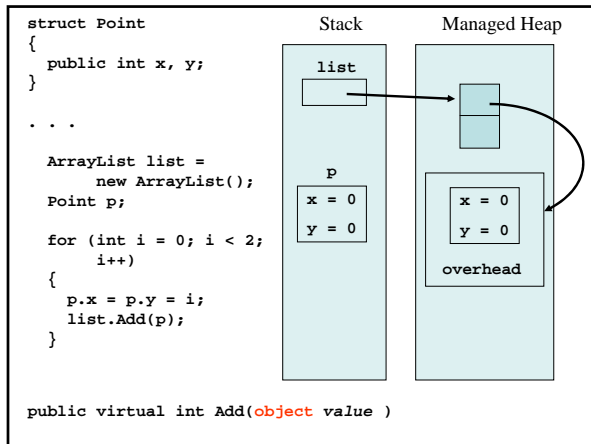
struct Point
{
  public int x, y;
}

. . .

ArrayList list =
  new ArrayList();
Point p;

for (int i = 0; i < 2; i++)
{
  p.x = p.y = i;
  list.Add(p);
}
    
```





```

C#
Stack s = new Stack();
s.Push(19);
s.Push(28);
int x = (int) s.Pop();
int y = (int) s.Pop();
    
```

```

C#
Stack s = new Stack();
s.Push(19);
s.Push(28);
int x = (int) s.Pop();
int y = (int) s.Pop();

Java
Stack s = new Stack();
s.Push(new Integer(19));
s.Push(new Integer(28));
int x = ((Integer) s.Pop()).intValue();
int y = ((Integer) s.Pop()).intValue();
    
```

```

C#
Stack s = new Stack();
s.Push(19);
s.Push(28);
int x = (int) s.Pop();
int y = (int) s.Pop();

foreach (int z in s)
{
    Console.WriteLine(z);
}
    
```